Disseny i implementació d'una API REST per a la plataforma SIMBA

Henoc Martí Guerrero

Resum– Actualment, són moltes les aplicacions i projectes que disposen d'una API REST. De la mateixa manera, són moltes les empreses que generen negoci gràcies a les seves prestacions. Amb aquest tipus d'arquitectura s'obtenen avantatges tan importants com el creixement horitzontal i la programació d'API's més eficients i enfocades a la lògica d'Internet. El SIMBA és la plataforma integral d'indicadors metropolitans de l'IERMB. Proporciona de forma pública l'accés a una informació precisa, actualitzada i comparable en forma d'indicadors que permeten el seguiment en el temps i l'espai. Aquesta web d'estadística s'ha quedat desfasada: disposa de poca escalabilitat, poca flexibilitat i un consum de recursos elevat. D'aquesta necessitat neix el projecte actual: dissenyar i implementar un nou back-end amb arquitectura REST capaç de gestionar eficientment les consultes sobre els indicadors en formats estàndard.

Paraules clau– API, REST, HTTP, PHP, Monolog, JSON, JWT, PostrgreSQL, Apache, back-end

Abstract– Nowadays, there are many apps and projects that have an API REST available. Likewise, there are many companies which are operative thanks to them. With this type of structure, they obtain advantages such as horizontal expansion and the most efficient and focused on Internet's logic API's programming. SIMBA is the integral platform of IERMB's metropolitan indicators. It provides public access to precise and updated information, which can be compared through indicators that allow its following through time and space. This statistics website has become outdated: it only has a short range, minimum flexibility and a high cost of operation. From this necessity this project was born: designing and implementing a new back-end with REST structure capable of managing eficiently the standard indicators enquiries.

Keywords– API, REST, HTTP, PHP, Monolog, JSON, JWT, PostrgreSQL, Apache, back-end

1 Introducció

L'INSTITUT d'Estudis Regionals i Metropolitans de Barcelona (IERMB) [1] és un consorci públic adscrit a l'Àrea Metropolitana de Barcelona que està situat al campus de la Universitat Autònoma de Barcelona. L'objectiu d'aquest organisme és desenvolupar activitats de recerca, formació i difusió en l'àmbit urbà, social, ambiental, econòmic i territorial de l'àrea i la regió metropolitana de Barcelona. L'IERMB s'encarrega del manteniment i desenvolupament de la web Sistema d'Indicadors Metropolitans de Barcelona (SIMBA) [2], una plataforma estadística que ofereix públicament un conjunt exhaustiu d'indicadors metropolitans amb continguts, per exemple, de demografia, cohesió social, habitatge, convivència i seguretat, economia i mobilitat. El sistema permet visualitzar, filtrar i descarregar aquesta informació en funció de paràmetres molt variats. Aquests indicadors metropolitans són utilitzats per institucions públiques, investigadors i empreses, entre altres, per realitzar estudis de mercat i presa de decisions. Aquesta plataforma, que es va crear l'any 2012 i actualitzar el 2014, es va desenvolupar amb un disseny poc flexible i amb les tecnologies d'aleshores (actualment opera amb versions de programari desactualitzades). El programari del servidor, tot hi que continua operatiu, ha quedat desfasat a causa del seu disseny poc modular i arcaic. Pels administradors de la web se'ls ha tornat complicadíssim millorar-ne les prestacions o implementar-hi noves funcionalitats. És per aquest motiu que l'IERMB ha iniciat un projecte de renovació de tota la plataforma dividint-la amb dues etapes:

1. Disseny i implementació del back-end [3]: desenvolupament del programari que processarà les consultes a la base de dades.

• E-mail de contacte: henoc.marti@e-campus.uab.cat
• Menció realitzada: Tecnologies de la Informació
• Treball tutoritzat per: Ramon Martí Escalé (Enginyeria de la Informació i de les Comunicacions)
• Curs 2018/19

2. Disseny i implementació del front-end: interfície de la web.

Aquest document descriu detalladament el desenvolupament de la primera etapa, un nou back-end per a la plataforma SIMBA. El nou motor de processament, basant-se en l'arquitectura de programari Representational State Transfer (REST) [4], està compost per un conjunt de components i aplicacions que permeten gestionar eficientment les consultes al sistema d'indicadors. El resultat és la creació d'una Interfície de Programació d'Aplicacions (API) [5] REST, un servei web que permetrà obtenir dades o generar operacions sobre els indicadors en formats estàndard.

El projecte consta de diverses etapes. D'entrada es descriu el problema a resoldre, és a dir, juntament amb una breu contextualització de l'institut i la plataforma, es defineix breument la proposta del projecte. A continuació hi ha la definició dels objectius del projecte; aquest apartat descriu cada un dels objectius i subobjectius necessaris per desenvolupar correctament el nou servidor. Posteriorment hi ha l'estat de l'art, una secció que conté la base teòrica que sustenta l'escrit i les tecnologies utilitzades. La metodologia utilitzada, que s'esmenta al seu corresponent apartat, és la tècnica de Refactoring. A continuació hi ha la planificació, un apartat que conté una descripció de totes les tasques a realitzar juntament amb els recursos necessaris. Posteriorment hi ha la documentació associada al desenvolupament de cada una de les tasques estructurada d'acord amb la metodologia (patró circular). Finalment, hi ha l'exposició dels resultats i les conclusions.

2 OBJECTIUS

L'objectiu general del TFG és la creació d'un nou motor de processament (back-end) de la versió 2 de SIMBA que defineixi un servei web (API) basat en l'arquitectura REST capaç de proporcionar i gestionar consultes als indicadors de l'IERMB.

La complexitat del sistema d'Indicadors de la plataforma és alta, és a dir, el back-end ha de ser un programari adaptat a la naturalesa de cada indicador. Podem diferenciar dos tipus de funcionalitats "independents" i globals: consultes a les categories i consultes als indicadors. Aquestes dues funcionalitats que són interdependents en l'àmbit operatiu, seran implementades a través de dues aplicacions diferents. Per desenvolupar la plataforma ens basarem en el disseny estructurat d'aplicacions i mòduls, és a dir, en la programació modular.

Els objectius, ordenats en funció de la seva prioritat de realització, són els següents:

1. **Anàlisi i documentació de Requisits**: definició del sistema a desenvolupar a partir dels requeriments generals de l'IERMB i de la plataforma actual. Per definir com ha d'interactuar el sistema, s'ha utilitzat la tècnica de casos d'ús. Proporcionarà una visió clara de com reacciona el sistema davant possibles esdeveniments. Permetrà enfocar les necessitats reals que l'usuari necessita facilitant la priorització del requisit.

2. **Disseny dels mòduls del back-end**: disseny global de la plataforma. En base aquest disseny, s'ha documentat de cada un dels blocs lògics (mòduls) la seva funcionalitat, el seu abast i els recursos necessaris.

3. **Desenvolupament dels mòduls de suport**: creació dels mòduls de suport de l'API d'acord amb la metodologia del projecte. Aquests mòduls proporcionaran funcionalitats específiques que les llibreries i aplicacions utilitzaran sense arribar a formar-hi part. Els mòduls de suport són:

   - Llibreria Logger Adapter: llibreria que permet definir una interfície de mètodes globals per gestionar els registres. Adaptarà llibreries de Logger existents a la nostra interfície.
   - Error Handler: mòdul que té per objectiu gestionar els errors.
   - Session: secció lògica de la plataforma que regula les sessions d'usuari i dades persistents de sessió.
   - File Handler: mòdul que gestiona la lectura de paràmetres de fitxers CSV [6].
   - Database: secció lògica de la plataforma que conté tots els mètodes per realitzar consultes a la base de dades.

4. **Desenvolupament de la llibreria API**: llibreria que té per objectiu definir una API per a les aplicacions Indicador i Categoria. Aquesta llibreria també proporciona els mètodes i procediments necessaris per gestionar els paràmetres d'entrada de les aplicacions.

5. **Desenvolupament de l'aplicació Category**: aplicació que permet obtenir la informació relativa a les categories dels indicadors en funció de certs criteris de filtratge. Aquests criteris, que s'especificaran a la URI, permetran obtenir per exemple les subcategories d'una categoria concreta, el màxim nombre de nivells o l'idioma.

6. **Desenvolupament de l'aplicació Indicator**: aplicació que permet obtenir la informació corresponent als diferents indicadors en funció de l'àmbit, la temporalitat, l'idioma o de la/les variable/s independent/s. La complexitat de processament d'aquesta és alta perquè ha de permetre accions tan variades com per exemple l'agregació de variables.

7. **Documentació del projecte**: documentació global del projecte on també s'hi inclou els objectius i requisits. Hi haurà les especificacions tècniques de cada un dels mòduls del back-end amb els seus respectius UML [7] i manuals d'utilització.

3 ESTAT DE L'ART

Aquest apartat descriu la plataforma actual i les tècniques REST i Monolog utilitzades per al desenvolupament d'aquest projecte.

3.1 Plataforma SIMBA

SIMBA és una plataforma web de l'IERMB desenvolupada principalment amb PHP [8] (llenguatge de programació del motor de processament), HTML, JavaScript i SQL. Els recursos es troben allotjats a un servidor virtual Debian amb HTTP Apache [9] dins el domini de la UAB.

La pàgina web disposa d'una interfície d'usuari amb tres funcionalitats bàsiques. La primera consisteix a proporcionar una estructura de categories per organitzar i poder escollir eficientment els indicadors. Estan organitzats en 8 categories globals i un conjunt divers de subcategories en funció de la temàtica. La segona funcionalitat és l'execució i presentació de la consulta d'un indicador. Permet executar les consultes en funció de paràmetres de filtratge (variables de l'indicador) i mostrar els resultats en taules o gràfics. També dona la possibilitat de descarregar els resultats. L'enviament de paràmetres al servidor es realitza mitjançant els mètodes GET i POST d'HTTP [10]. La tercera funcionalitat és un sistema de registre que permet obtenir privilegis per visualitzar indicadors protegits o modificar el contingut.

L'arquitectura de la web és confusa, la capa de presentació (front-end) i la d'accés a dades (back-end) estan solapades, no hi ha abstracció ni capa de persistència. Tampoc no existeix un sistema de proves per comprovar prèviament canvis i el sistema de registre és ineficient. Les dades estan emmagatzemades en una base de dades relacional PostgreSQL [11] i totes estan en català.

3.2 REST

REST és una arquitectura de software per sistemes d'hipermèdia distribuïts. Aquesta tecnologia utilitza el protocol HTTP per executar operacions sobre dades o consultar recursos en diferents formats. Les característiques més importants [12] que defineixen aquesta tecnologia són les següents:

- Protocol client/servidor sense estat: cada petició conté totes les dades necessàries per processar la consulta, el servidor no necessita emmagatzemar cap informació de l'usuari.
- Identificació de recursos a partir de la URI [13]: els objectes REST es manipulen mitjançant un identificador únic, la URI.
- Sistema de capes: arquitectura jeràrquica entre components.

3.3 Monolog

Monolog [14] és una llibreria amb llicència del MIT que permet gestionar eficientment el registre d'esdeveniments (logs). Està dissenyada d'acord als estàndards PSR-3 (Proposed-standard-request) [15] de PHP. Algunes de les operacions que permet, per exemple, són enviar o emmagatzemar els registres a fitxers, al correu electrònic, a bases de dades o a diversos serveis web. Gràcies a controladors especials, és possible construir estratègies de registre avançades.

4 METODOLOGIA

Per desenvolupar un software de qualitat, s'ha decidir utilitzar la tècnica Refactoring [16]. Consisteix a seguir un patró circular: estudi dels requisits i anàlisi, disseny (UML), implementació/desenvolupament i proves de test. Aquesta metodologia s'aplica a cada una de les activitats descrites a l'apartat anterior (excepte les activitats que no són de programació: 1, 2 i 7).

Al llarg del projecte:

- S'han realitzat reunions setmanalment per tractar l'estat de l'art, ajustar la planificació, solucionar incidències, etcètera.
- En cas de dubtes i/o problemes, l'acció estandarditzada ha sigut la contrastació d'idees, és a dir, la discussió del tema amb els companys de l'empresa amb l'objectiu de prendre la millor decisió possible.

5 DESENVOLUPAMENT

El progrés del projecte ha estat assolit correctament seguint la jerarquia d'etapes i terminis temporals especificats a la planificació inicial del projecte.

La metodologia de treball descrita a l'apartat anterior s'ha executat satisfactòriament. Ha estat fonamental per assolir eficientment la presa de decisions i ha permès establir un canal de comunicació constant i fluid. La tècnica de Refactoring ha estat essencial durant el desenvolupament per obtenir un codi de qualitat, és a dir, un programari amb un codi fàcil de comprendre i amb un disseny flexible i eficient que facilitarà el manteniment en un futur. També s'han dissenyat i implementat proves de test per millorar la robustesa i la fiabilitat.

A continuació s'especifica tot el treball realitzat per cada un dels 7 objectius del projecte.

5.1 Anàlisi i documentació de requisits

L'anàlisi i documentació de requisits s'ha complementat amb la col·laboració de l'IERMB. S'ha documentat cada una de les necessitats singulars que ha de permetre realitzar l'API de plataforma SIMBA, és a dir, quins serveis ha d'oferir i com ho farà. Els requisits serveixen com base a l'hora de dissenyar el projecte, però també són igual d'importants i necessaris durant el procés de validació.

La confecció de requisits s'ha dut a terme d'acord amb l'anàlisi de la plataforma actual i de les necessitats dels usuaris (IERMB). S'han acceptat 7 requisits globals per l'API SIMBA:

1. La plataforma ha de gestionar consultes a dos tipus de recursos: a categories i indicadors. S'haurà de gestionar paràmetres d'usuari i variables d'entorn. El format estàndard de sortida serà JSON.
2. L'API SIMBA gestionarà les consultes a:
    - Un indicador amb àmbits i temporalitat.
    - Un indicador amb àmbits, temporalitat i variable independent.

- Un indicador amb àmbits, temporalitat, variable independent i/o variable independent agregable.

3. Les consultes mostraran les dades en la seva totalitat, és a dir, si no hi ha restriccions paramètriques o de privilegis, el recurs obtingut serà total.
4. El sistema disposarà de procediments i mètodes per gestionar l'accés i l'escalada de privilegis.
5. Sense considerar els paràmetres de cada aplicació, totes hauran de poder gestionar peticions en funció de l'idioma.
6. La plataforma gestionarà eficientment els errors.
7. La plataforma disposarà d'un sistema de registre per registrar les activitats dels usuaris, errors, etcètera.

Dels requisits especificats, s'han definit respectivament els seus casos d'ús. La descripció de cada una de les activitats proporcionaran una visió clara de com reacciona aquesta davant els possibles esdeveniments. Tanmateix, també permet enfocar les necessitats reals que l'usuari necessita facilitant la priorització del requisit. El contingut de cada cas d'ús és complet. Hi ha una breu descripció de la funcionalitat del requeriment, es defineixen les precondicions, seqüència lògica i postcondicions del recurs. També conté excepcions i comentaris que complementen amb informació substancial el requisit.

Per definir correctament les consultes REST, hi ha un apartat dins la documentació de requisits on es descriu les possibles consultes que permetrà l'API SIMBA. A cada una d'elles s'especifica els paràmetres d'entrada i el format de sortida exactes. Els paràmetres de la consulta podran enviar-se al servidor (back-end) a través dels mètodes GET o POST. Per realitzar una consulta a una aplicació o una altra, el nom de l'API s'especificarà a la URI.

5.2 Disseny dels mòduls del back-end

Aquesta és l'etapa del projecte té per objectiu dissenyar el sistema global de l'API. Bàsicament consisteix en analitzar els requisits, proposar una estructura lògica de classes i llibreries per satisfer els serveis i comparar aquesta solució amb altres possibilitats.

El disseny del back-end està confeccionat d'acord amb tres objectius: ha de ser escalable, optimitzat i flexible. Per assolir aquests objectius, el back-end estarà format per diferents mòduls lògics (encapsulació) on cada un proporciona una funcionalitat específica (Figura 1).

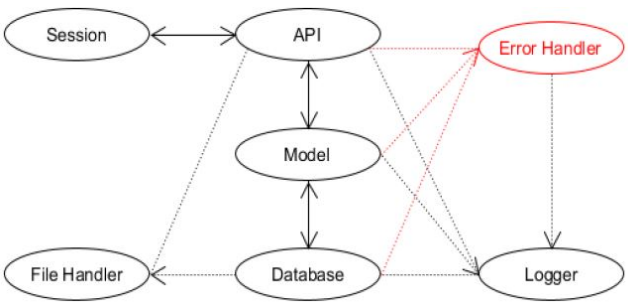

Fig. 1: Estructura dels mòduls de la plataforma

La definició de cada un dels mòduls és la següent:

- API: llibreria que gestiona els paràmetres de la consulta, crea l'aplicació adient, controla l'accés a les funcionalitats i retorna el recurs sol·licitat (requeriments 1 i 4).
- Model: hi ha definides les aplicacions Category i Indicator (reqs. 2, 3 i 5).
- Database: conté els mètodes per realitzar les consultes a la base de dades.
- Session: mòdul que gestiona paràmetres de sessió de l'usuari a la memòria al servidor (req. 4).
- Logger: llibreria que registra l'activitat (req. 7).
- File Handler: mòdul que permet obtenir les dades de fitxers externs.
- Error Hanlder: gestiona els errors (req. 6).

La programació de tots aquests mòduls es realitzarà bàsicament amb llenguatge PHP i SQL utilitzant com a entorn de treball l'Eclipse [17] i PostrgreSQL.

5.3 Mòduls de suport

El desenvolupament dels mòduls de suport s'ha diferenciat en diferents etapes/subobjectius (d'acord amb la funcionalitat).

5.3.1 Llibreria Logger Adapter

Aquest mòdul, anomenat Logger Adapter, té com a objectiu adaptar un sistema de registre ja existent a la interfície de logging personalitzada.

Anàlisi

La llibreria Logger Adapter permet definir una sèrie d'operacions bàsiques que per tots els sistemes de logging que s'utilitzin, per molt diferents que siguin, realitzaran la mateixa acció. D'aquesta manera, amb aquest mòdul d'adaptació, s'utilitzaran les declaracions definides a la interfície per afegir un log aconseguint així no dependre de les funcions pròpies de cada un d'ells. Aquesta funcionalitat, que és l'objectiu bàsic d'aquesta llibreria, serà de gran utilitat per no haver de dependre d'una llibreria de registre en concret i permetrà que la gestió de registres a la plataforma s'utilitzi de la mateixa manera independentment del sistema de registre natiu.

El paper que juga aquest mòdul lògic a la nostra plataforma és proporcionar un sistema de gestió i creació de registres. Emparant la llibreria nativa Monolog, la nostra llibreria permet afegir una capa d'abstracció i crear un programari de gestió de log's. Els registres que s'afegeixin es categoritzaran en nivells d'acord amb l'estàndard PSR-3.

Disseny

Per assolir aquest objectiu, aquesta llibreria ha estat dissenyada en base al patró de disseny Factory [18]. Permet gestionar tants logger's diferents com sigui necessari. Per si sol, si no s'afegeix cap logger a la llibreria (no hi ha logger's implementats a la factoria), no té funcionalitat. El disseny intern d'aquest mòdul en llenguatge UML es pot observar a la Figura 3 (apèndix A.1).

Com podem observar al diagrama, s'ha definit una interfície global anomenada *LoggerAdapterInterface* que defineix les operacions possibles d'aquest mòdul. L'objectiu d'implementar aquest disseny és definir una interfície d'adaptació (*HandlerLogInterface*) on poder implementar qualsevol llibreria de registre externa. Per assolir aquest objectiu, la factoria *HandlerLogFactory* crearà els objectes de tipus logger que siguin necessaris i estiguin definits. Tal com s'observa al diagrama, aquesta interfície de loggers serà implementada per la llibreria Monolog donant com a resultat dos gestors de registre diferents (MonologLoggerStream i MonologRotatingFileHandler).

Implementació
El desenvolupament d'aquest mòdul ha estat implementar el disseny prèviament definit. A partir de la definició de la Interfície Logger, s'ha implementat Monolog. Amb aquesta nova llibreria, s'han programat dos gestors de logs diferents per la plataforma. El primer, MonologLoggerStream, guarda tots els registres en un únic fitxer. El segon, MonologRotatingFileHandler, guarda els logs en fitxers diferenciats, és a dir, un per dia.

Test
L'últim a desenvolupar d'aquest mòdul són els tests. En aquesta etapa s'ha dissenyat i implementat amb PHPUnit [19] proves unitàries per verificar la funcionalitat de la llibreria LoggerAdapter. S'ha creat per a cada objecte el seu respectiu TestCase. D'aquesta manera, quan s'implementi un nova funcionalitat o s'actualitzi, els test permetran verificar que continua funcionant correctament.

5.3.2 Error Handler

Aquest mòdul gestiona possibles errors o accions irregulars que succeeixin durant l'execució d'una consulta a qualsevol mòdul o llibreria.

Anàlisi
Els errors que es puguin produir durant l'execució d'una consulta poden ser variats. En diferenciem dos tipus:

1. Controlats: errors específics o esdeveniments no desitjats que es gestionen a través de procediments injectats al codi. El programador implementa específicament aquests procediments als llocs concrets.

2. No controlats: errors que es produeixen per causes desconegudes. No hi ha cap procediment específic del programador per gestionar-lo (ho fa automàticament PHP).

Quan s'executi un error o una acció no permesa, aquest mòdul "matarà" l'execució de l'aplicació, assignarà un codi d'error a la resposta i capçalera de PHP, i retornarà un missatge d'error JSON. També afegirà un registre (usant la llibreria Logger Adapter) amb la informació de l'incident.

Disseny
D'acord amb aquests tipus d'errors, diferenciem dos seccions lògiques de gestió d'errors dins aquest mòdul. Per assolir el primer objectiu (errors controlats), i tenint en compte que aquesta funcionalitat ha d'estar present a qualsevol secció de la plataforma, s'ha dissenyat en base al patró de disseny Singleton [20]. Per gestionar el segon tipus d'errors, els no controlats, s'utilitzen dos mètodes natius de PHP que permeten definir la pròpia gestió dels errors en temps d'execució. S'utilitza una tercer mètode natiu de PHP que permet establir quins errors de PHP es notificaran.

Implementació
La implementació d'aquest mòdul ha estat diferenciada entre els dos tipus d'errors que es produeixen. La secció de processament de l'error controlat, formada per classes que implementen un interfície Singleton amb els mètodes bàsics de gestió i la secció de gestió d'errors no controlats. Aquest mòdul doncs, el formen dos seccions diferenciades que cooperen conjuntament.

Test
L'última etapa de desenvolupament d'aquest mòdul ha sigut programar proves amb PHPUnit.

5.3.3 Session

Aquest mòdul té per objectiu encapsular els paràmetres de sessió de l'usuari. Tot i que l'API REST és Stateless, aquest mòdul permet gestionar eficientment dades úniques de sessió.

Anàlisi
Gestionar les dades d'usuari durant la mateixa connexió és imprescindible per obtenir estadístiques d'ús unitàries i monitoritzar accions sospitoses o no desitjades. Les funcionalitats són les següents:

- Emmagatzematge de la instancia de l'objecte "Logger Adapter". Evitem així crear objectes de registre cada cop que s'executa una consulta.

- Gestió d'una identificació única per a cada sessió. Aquest identificador serà necessari per diferenciar els registres de seguiment, errors, etcètera.

- Gestió de privilegis i credencials.

Disseny
L'objecte *Session* està dissenyat d'acord al patró Singleton. D'aquesta manera és accessible a qualsevol secció del programari i és molt fàcil d'instanciar. Per seguretat, les sessions tenen un temps limitat, és a dir, si no hi ha activitat en un cert temps, expiren.

Implementació
La implementació de les funcionalitats descrites a l'apartat d'anàlisi, s'han implementat utilitzant el mètode de gestió de sessions de PHP i la variable superglobal *$_SESSION*. Utilitzant aquests gestors s'ha programat un objecte singleton que defineix, inicialitza, gestiona i destrueix les dades que s'emmagatzemen a la variable superglobal de sessió.

Test
Per verificar el correcte funcionament d'aquest mòdul, s'han creat TestCase per a cada objecte. Com que aquest mòdul opera amb variables de sessió, s'han dissenyat i implementat proves de test que gestionen diferents valors a

la variable superglobal *$_SESSION* permetent verificar els mètodes.

5.3.4 File Handler

Aquest mòdul encapsula els procediments necessaris per llegir fitxers CSV.

Anàlisi
L'objectiu és obtenir un component de la plataforma que gestioni la lectura de fitxers amb dades separades per un delimitador. En general, l'estructura de dades dels fitxers CSV a llegir serà bidimensional, de Clau-Valor. Els usos són variats.

Disseny
El disseny d'aquest mòdul és molt simple. Està format per una única classe que defineix una sèrie de mètodes per gestionar la lectura i obtenció de dades. L'obtenció dels valors del fitxer pot ser selectiva, és a dir, a partir de la clau, o pot ser completa (s'obtenen totes les dades del fitxer).

Implementació
La implementació s'ha realitzat d'acord al disseny. S'han incorporat elements de gestió d'errors i anàlisi (creació de logs).

Test
Les proves de test d'aquest mòdul s'han desenvolupat correctament amb el framework PHPUnit. S'han dissenyat i implementat mètodes per comprovar que la lectura de fitxers fos correcte.

5.3.5 Database

Aquest mòdul gestiona les consultes a la base de dades, és a dir, separa les instruccions SQL de les aplicacions.

Anàlisi
Té per objectiu encapsular tots els procediments relacionats amb l'obtenció de dades a la base de dades i proporcionar els mètodes públics justos i necessaris per poder obtenir les dades que requereixi cada aplicació tenint en compte la varietat d'operacions i necessitats de cada una.

Disseny
L'estructura d'aquest sector lògic està dissenyada per ser escalable a altres sistemes de gestió i ser flexible amb les necessitats de les API's implementades. Per assolir aquests objectius, s'utilitza una interfície global per implementar tots els procediments de gestió amb els mateixos mètodes i el principi de l'herència d'objectes per modelar els sistemes implementats en funció de les necessitats de cada API. És doncs, un mòdul que proporciona una capa d'abstracció a l'accés de dades. La interfície és independent de la base de dades que s'utilitzi.

Implementació
Aquest mòdul s'anirà implementant en la seva totalitat a mesura que es desenvolupin les aplicacions Category i Indicator. D'entrada s'implementen els mètodes necessaris per establir la connexió a la base de dades PostgreSQL i gestionar les consultes i errors.

Per a cada aplicació es definirà una abstracció nova, és a dir, una instància única que diferenciarà les diferents funcionalitats. Els mètodes estaran definits en base als paràmetres necessaris de cada consulta. Per exemple, una de les noves característiques de la nova web, és que els recursos estaran traduïts a diferents idiomes. A part de les abstraccions de cada aplicació, hi ha una objecte que s'estén de la interfície, *PGStatement*. Aquesta classe permet injectar les parts d'una consulta SQL per separat amb l'objectiu d'executar-la posteriorment. Aquest objecte no està associat a cap aplicació però s'utilitza a l'Indicator.

Test
Les proves de test amb PHPUnit d'aquest mòdul són únicament de la capa de connexió amb la base de dades PostgreSQL. Aquests procediments comproven que la connexió PDO sigui correcta. Es fan proves singulars d'obtenció de dades per comprovar el correcte funcionament.

5.4 Llibreria API

Aquesta llibreria té com a objectiu crear les aplicacions necessàries per processar la consulta del recurs i validar els paràmetres d'entrada.

Anàlisi
Aquest programari proporciona un servei: crear aplicacions. La llibreria API té les següents funcionalitats:

- Creació de les aplicacions Category i Indicator.
- Gestió dels paràmetres de les aplicacions: a part d'obtenir-los, conté procediments per gestionar-los efectivament i validar-los.
- Validació del tipus de mètode utilitzat per accedir al recurs. Permet: GET i POST.
- Integració d'una API alternativa (patró d'herència) per si es vol realitzar autenticació per token.
- Emmagatzematge dels mètodes necessaris per gestionar errors i guardar l'activitat (logging).

Disseny
Aquesta API especifica una interfície de programació a través d'una classe abstracta. Tot aquest conjunt de funcions i procediments, defineixen el contracte amb els controladors (model), és a dir, dona suport a les invocacions a serveis fetes per altres aplicacions (en el nostre cas, l'aplicació Indicator i Category). Quan una aplicació implementa aquesta interfície (proveeix de funcionalitat) diem que és una implementació de l'API. Per gestionar les aplicacions de l'API, el disseny lògic està basat en el patró "factoria".

Cada aplicació té un diccionari propi que conté les propietats i característiques que han de tenir els paràmetres permesos. Aquest diccionari, que es carregarà durant la inicialització de l'aplicació, permetrà a la llibreria API gestionar les dades d'entrada.

Els paràmetres de la consulta poden enviar-se al servidor (back-end) a través dels mètodes GET o POST, en cas contrari, es mostrarà un missatge d'error. Per realitzar una consulta a una aplicació o una altra, el nom de l'API s'especificarà a la URI (Figura 2).

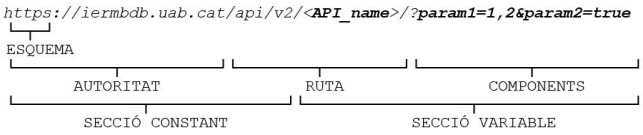

Fig. 2: Format de la URI

En temes de seguretat hi ha certes restriccions, l'ús de la plataforma estarà restringit. Per gestionar l'accés sense haver de validar les credencials cada vegada, es crearà una Cookie HTTPOnly [21] amb un JSON Web Token (JWT) de sessió. El procediment per inicialitzar aquest token de sessió serà de dues maneres diferents: a partir d'un token JWT que l'usuari enviarà com un paràmetre o validant-se a través de credencials convencionals. La validació per token té certs avantatges sobre altres tipus d'autentificació que la fan ideal per una API REST: no és necessari que el servidor guardi cap mena d'informació de l'usuari (stateless), només cal guardar la clau per signar els tokens. Aquesta clau privada s'emmagatzemarà en un fitxer al servidor. Per realitzar la creació i validació de tokens utilitzem l'estàndard JWT, una llibreria amb llicència del MIT capaç de generar i validar tokens amb una àmplia gamma d'algoritmes.

Tal com es representa al diagrama UML, Figura 4 (apèndix A.1), la classe principal *ApiRest* instancia a les classes *HandlerUrl* per gestionar els paràmetres i *HandlerApiFactory* per gestionar les aplicacions. La factoria crea les aplicacions que s'hagin implementat a partir de la interfície *ApiInterface*. Per gestionar els paràmetres dins les aplicacions, hi ha l'objecte *ListParams* que permet organitzar-los un cop s'han validat.

Implementació
La implementació ha estat efectuada d'acord al disseny intern de la llibreria. Observem com a partir de l'objecte principal *ApiRest*, es gestiona la factoria de les aplicacions Category i Indicator. Per implementar la funcionalitat de validació JWT s'ha desenvolupat un submòdul apart anomenat AccesControl que gestióna l'accés a la llibreria (autentica als usuaris i administra les credencials).

Test
Les proves de test d'aquest mòdul, dissenyades i implementades amb PHPUnit, testegen cada un dels mètodes de les classes. D'aquesta manera, comprovant la funcionalitat unitària de cada funció, es controla el bon funcionament ara i en futures versions.

5.5 Aplicació Category

L'aplicació Category s'encarrega de gestionar les consultes a recursos relacionats amb les categories de la base de dades.

Anàlisi
L'aplicació, lògicament parlant, la situem al Model i/o Controlador (arquitectura MVC). Conté la lògica dels negocis, és a dir, hi ha definits els objectes i mètodes que es comuniquen directament amb el mòdul DataBase però també s'encarrega de retornar les dades del recurs.

Disseny
El processament de les dades consultades i el lliurament del recurs estan dissenyats i implementats en base a les definicions dels requisits i restriccions. Per assolir aquest objectiu, es disposa d'una estructura de classes que permet:

1. Organitzar l'estructura del recurs (les categories i les seves dades) en funció del grau de jerarquia/nivell (subcategories). Ordre taxonòmic.

2. Retornar el recurs en format JSON.

Per organitzar les categories, aquest mòdul conté els procediments per estructurar les dades de consulta la consulta. Amb els valors obtinguts (una matriu multidimensional) es construeix una estructura de nodes. L'objectiu és crear una estructura que permeti organitzar jeràrquicament les categories. Per aconseguir-ho, es crea un graf acíclic connex (arbre) amb un node principal (root) que conté tot de nodes (categories) amb els seus respectius fills (subcategories). A partir d'aquesta estructura, es converteix l'arbre de categories (la informació dels nodes) a una cadena de text en format JSON. La serialització de les dades es realitza mantenint l'ordre de dependències: el resultat final és una cadena amb les categories i subcategories ordenades jeràrquicament.

El disseny intern d'aquesta aplicació, d'acord amb la interfície d'aplicació de la llibreria API, el podem observar a la Figura 5 (apèndix A.1). La classe *ControllerCategories* implementa els mètodes de la interfície de programació de la llibreria API. Les classes *LoadCategories* i *Category* gestionen l'obtenció i serialització de les dades, mentre que la classe *AdapterCategoryJSON* genera el la sortida JSON.

Implementació
Com es pot observar a la Figura 5 (apèndix A.1), les aplicacions hereten de la interfície d'aplicacions definida a la llibreria API. L'aplicació Category és doncs, la implementació de la interfície (API) juntament amb els objectes necessaris per gestionar les consultes. Els objectes principals d'aquesta aplicació són *LoadCategories*, que té permet obtenir les dades i organitzar-les (estructura d'arbre) i l'objecte *AdapterCategoryJSON*, que genera el recurs final. El diccionari, que s'utilitza per gestionar els paràmetres, està definit en un fitxer PHP i conté una matriu de caràcter constant on hi ha definides les propietats dels paràmetres acceptats.

Test
L'última etapa d'aquest mòdul, seguint la metodologia de treball, ha sigut el disseny i implementació de les proves de test unitàries. Igual que la llibreria API REST, aquest mòdul s'ha testejat completament, associant a cada objecte TestCase una classe de la llibreria. El resultat són un conjunt de procediments executables que permeten comprovar la funcionalitat dels mètodes. A part de les proves de test, s'han executat consultes a través del navegador, tal hi com ho faria l'usuari, per verificar el correcte funcionament.

5.6 Aplicació Indicator

Aquest mòdul té per finalitat definir i implementar els mètodes i procediments necessaris per assolir els objectius de l'aplicació Indicator i acomplir els requisits.

Anàlisi
L'objectiu d'aquest mòdul és processar de forma eficient els paràmetres de la consulta per obtenir un recurs optimitzat de les metadades, les variables i els valors dels indicadors. Obtenir les dades, filtrar-les i modelar-les forma part de les consultes que realitzarà aquesta llibreria al mòdul de la base de dades.

Els indicadors, que estan categoritzats (funcionalitat de l'aplicació Category), estan emmagatzemats en una base de dades relacional i s'identifiquen de forma numèrica amb un codi únic. Cada indicador, en general, disposa de dues taules. A la primera s'hi defineixen les metadades, informació general de l'indicador. A la segona hi ha els valors de l'indicador en funció de les variables.

Disseny
És una aplicació basada en la interfície d'aplicació de la llibreria API. Aquesta aplicació ha de gestionar les consultes que l'usuari executi a l'API, unes consultes que degut a la varietat d'indicadors i tractaments individualitzats, són complexes i difícils de dissenyar en un sistema modular eficient. Igual que l'aplicació Category, Indicator també disposa del seu diccionari de caràcter constant per validar els paràmetres d'usuari. L'estructura del recurs final JSON, que s'especifica a continuació, està formada per tres apartats.

- Metadades: secció que conté una llista amb la informació de l'indicador, com per exemple el títol, la descripció, les unitats, etcètera.
- Variables: secció que conté una llista de les variables que defineixen l'indicador. Aquest llistat relaciona l'identificador de la variable amb el nom que té (diccionari de variables).
- Data: secció que conté els valors de la consulta, és a dir, el resultat. Cada valor està associat a una matriu amb les variables que el defineixen. Les variables estan ordenades d'acord amb l'ordre dels camps a la base de dades.

Considerant que les dades que s'han obtenir es poden classificar en tres seccions, metadades, variables i dades, l'estructura que s'ha dissenyat per aquest mòdul és la següent. S'ha dissenyat un objecte anomenat *Indicator* que encapsula totes les dades del recurs, un objecte on a mesura que s'obtinguin aquestes dades s'injectaran. La classe que implementarà els mètodes per obtenir/retornar l'objecte *Indicator* omplert amb totes les dades s'anomena *LoadIndicator*. La lògica d'execució d'aquesta classe per assolir aquest objectiu és la següent:

1. Càrrega de les metadades de l'indicador (consulta a la base de dades a través del mòdul Database).
2. Creació dels objectes associats a cada variable: objectes que permetran gestionar de forma única les propietats de cada variable a les consultes (SQL).
3. Obtenció dels valors de la consulta.
4. Execució de la inicialització del diccionari de cada variable. Relaciona cada identificador d'una propietat d'una variable amb el seu nom.
5. Generació l'objecte *Indicator* a partir de l'identificador de l'indicador i la matriu de metadades, de variables i de valors. Finalment retorna l'objecte.

Com que les variables són un component bàsic dels indicadors molt divers, els procediments únics i variats en que es creen els filtres (per les consultes SQL) o s'omplen els diccionaris s'externalitzen a objectes. Així doncs, els objectes de tipus Variables permeten gestionar els diferents tipus de variables (els anys, els mesos, els àmbits, les variables independents, etcètera). Aquest objectes estan estructurats amb el patró de disseny factoria i implementen els mètodes que defineix la interfície de variable *VariableTypeInterface*. L'objectiu final d'aquests objectes de tipus variables és injectar els filtres de l'usuari i funcions SQL, gestionar les agregacions i omplir els diccionaris.

Com es pot observar al diagrama UML, Figura 6 (apèndix A.1), la classe *ControllerIndicator* s'estén de l'API i implementa els mètodes abstractes. Instancia la classe *LoadIndicator* per carregar les dades de la consulta a l'objecte *ndicator* i a la classe *FactAdaptOutInd* per gestionar el format de les dades. La factoria de variables es divideix en dues categories globals: les variables simples, formades per un única dimensió, i les variables bidimensionals, formades per dues dimensions. Hi ha variables que poden tenir comportaments diferents en funció de paràmetres contextuals, per aquest motiu, poden estar definides per dos o més classes (ex: *VarMonth* o *VarIndependent*).

Implementació
La implementació d'aquest mòdul s'ha completat amb èxit basant-se amb el diagrama UML dissenyat a la fase anterior. La implementació de la interfície API Rest s'ha programat satisfactòriament: el resultat de cridar el mètode getData() és l'obtenció del recurs sol·licitat. El desenvolupament de la factoria de variables ha permès organitzar la gran varietat de procediments que requereixen les diferents variables, procediments que de no haver estat implementats adequadament serien molt costosos.

Test
Per finalitzar, s'han dissenyat i implementat proves de test amb PHPUnit que verifiquen el correcte funcionament de les funcions de cada classe. A excepció dels mètodes que executen funcions del mòdul Database. Quan es creen les proves de test no només es verifiquen els mètodes, també es comproven el tipus d'atributs i el tipus d'objectes instanciats. En aquesta etapa també s'han executat consultes per verificar que l'aplicació funciona perfectament.

5.7 Documentació del projecte

L'última etapa del projecte és crear la documentació associada a tot el progrés, desenvolupament i funcionament de la plataforma. Per oferir correctament tota aquesta informació variada als usuaris, s'ha creat diferents documents.

5.7.1 Documentació general

Aquest document inclou la informació general del projecte, és a dir, conté el marc contextual, la definició i com s'ha desenvolupat. Alguns dels apartats més important són el

posicionament i interès del projecte dins l'IERMB, la contextualització tecnològica i l'anàlisi de requisits i definició dels objectius.

5.7.2 Documentació tècnica

Aquest document conté la informació tècnica del projecte. Es detalla el disseny de la plataforma i es descriuen tots els mòduls juntament amb els seus respectius objectes, mètodes, propietats, proves de test, etcètera. Aquesta documentació inclou una subsecció anomenada "Manual del programador" que te per objectiu assistir a "futurs" programadors a realitzar el desenvolupament i manteniment de la plataforma (afegir una nova llibreria al Logger Adapter, afegir una nova aplicació, canviar la configuració de paràmetres, etcètera).

5.7.3 Manual d'usuari

Aquest manual conté una guia escrita de com utilitzar la plataforma. Defineix com l'usuari pot executar les consultes a la plataforma, quines propietats tenen les aplicacions i com gestionar els paràmetres.

6 RESULTATS

El resultat d'aquest projecte ha sigut la creació d'un back-end per a la plataforma SIMBA en base a l'arquitectura de programació REST. S'ha aconseguit assolir tots els objectius. Amb el desenvolupament d'una llibreria API i un conjunt de mòduls de gestió, s'han creat dues aplicacions que permeten consultar un conjunt exhaustiu d'indicadors metropolitans mitjançant peticions web identificades a través de la URI. El format dels recursos obtinguts és JSON. A la Figura 7 i a la Figura 8 (apèndix A.2) s'hi pot observar, respectivament, el resultat d'una consulta de cada aplicació. El disseny optimitzat de la plataforma, la nova estructura de processament i la definició de consultes SQL eficients, ha reduït el temps d'execució de les consultes respecte la plataforma actual. Un objectiu de qualitat que s'ha assolit amb èxit ha estat el desenvolupament modular i escalable. El resultat és una plataforma totalment flexible que persegueix reduir el mínim de dependències entre llibreries, objectes, procediments, etcètera. Aquest servei web és fàcilment anàleg al context actual, són moltes les aplicacions i projectes que disposen d'una API REST. També són moltes les empreses que generen negoci gràcies a les seves prestacions. L'acompliment dels objectius planificats és el següent:

1. S'ha **analitzat i documentat els requisits** adequadament amb la suficient documentació per assentar les bases del projecte i acomplir els requeriments de l'IERMB.
2. El **disseny dels mòduls del back-end** ha estat essencial per estructurar la plataforma eficientment, organitzar les tasques i limitar el treball (no fer més feina de lo necessari).
3. S'ha desenvolupat els **mòduls de suport** d'acord amb la metodologia. Gràcies al disseny modular s'han independitzat funcionalitats de les llibreries i aplicacions programades.
4. S'ha implementat la **llibreria API** d'acord amb els requeriments proporcionant una interfície de programació per aplicacions per a la plataforma.
5. L'**aplicació Category** s'ha desenvolupat correctament obtenint com a resultat un programari capaç de gestionar les consultes a les categories dels indicadors.
6. S'ha desenvolupat l'**aplicació Indicator** amb un disseny eficient i modular capaç de gestionar la varietat de variables dels indicadors i processar les consultes de l'usuari.
7. Finalment la **documentació del projecte** ha estat una etapa on s'han creat varis documents que engloben tota la informació relativa al projecte.

Els coneixements associats al projecte són variats. Ha estat fonamental la capacitat individual per gestionar el temps i recursos, resoldre els problemes i tenir les habilitats de comunicació oral i escrita, capacitats que han estat nodrides durant els estudis del grau d'Enginyeria Informàtica i durant el desenvolupament d'aquesta plataforma. Ha estat bàsic disposar dels coneixements associats a les tecnologies de xarxa, web, multimèdia i seguretat per implementar un sistema de qualitat. Per exemple, els coneixements de tecnologies de xarxa han estat necessaris per desenvolupar la plataforma sobre el protocol HTTP i els coneixements de seguretat per desenvolupar un sistema d'accés segur a l'API REST. El coneixement base que considero més imprescindible és el disseny de patrons.

Finalment, d'acord amb els recursos monetaris i temporals, el desenvolupament d'aquest projecte de qualitat s'adapta a les perspectives esperades d'un estudiant universitari. Això ha estat possible gràcies a l'escalf de l'IERMB, el suport de constant del cap de departament i l'esforç invertit.

7 CONCLUSIONS

L'IERMB, a càrrec de la plataforma estadística SIMBA, ha iniciat un projecte d'actualització de la web per millorar les prestacions ja que s'ha quedat desfasada. El desenvolupament d'aquest projecte consisteix en dissenyar i implementar un nou back-end. El resultat ha estat una API REST que proporciona a l'usuari l'accés a dues aplicacions per gestionar les categories i dades dels indicadors de l'institut.

Aquest projecte s'ha desenvolupat d'acord a diferents objectius. D'entrada s'ha realitzat l'anàlisi i documentació inicial juntament amb el disseny dels mòduls del back-end. Posteriorment s'han desenvolupat els mòduls de suport i la llibreria API. S'han implementat dues aplicacions, la aplicació Category per gestionar les categories i l'aplicació Indicator per gestionar les de dades dels indicadors. Finalment s'ha documentat tot el projecte donant com a resultat diferents documents tècnics i manuals.

Gràcies a l'arquitectura REST i al disseny orientat a objectes, el nou back-end és escalable, flexible i eficient. Les aplicacions processaran les consultes a la base de dades registrant-ne les activitats, gestionant possibles errors i modelant el recurs final a format JSON. Aquestes qualitats han estat possibles gràcies a la metodologia establerta i a l'eficient planificació. La base del projecte s'assenta en l'anàlisi, documentació d'objectius i requisits i el disseny modular

de la plataforma. A partir d'aquest fonament, la realització dels objectius ha estat completa i exitosa, obtenint resultats molt més eficients a la plataforma actual.

Algunes possibles línies de continuació són la gestió de nous formats del recurs final i la creació d'una aplicació per tancar la sessió de l'usuari i una altre per crear credencials JWT. Una altre proposta és la creació d'un mòdul independent per gestionar eficientment les capçaleres HTTP. Amb aquest projecte desenvolupat, la segona etapa de desenvolupament de la plataforma SIMBA és la interfície web, és a dir, dissenyar i implementar el front-end d'acord amb l'API REST.

AGRAÏMENTS

Al Jordi Llobet, cap del projecte, pel seu constant suport, la seva visió crítica i la confiança dipositada en mi.

Al Ramon Martí Escalé, tutor del treball de fi de grau, per la seva exigència, la dedicació constant i l'orientació.

Al suport indirecte de l'entorn d'amistats, especialment a l'Anna i a l'Hector.

A tots els membres de la comunitat docent que, durant aquests anys, m'han nodrit amb els seus coneixements.

REFERÈNCIES

[1] Institut d'Estudis Regionals i Metropolitans de Barcelona, L'INSTITUT, 2019. [ONLINE]. Available: https://iermb.uab.cat/ca/linstitut/ [Accessed: 10- Mar- 2019]

[2] Institut d'Estudis Regionals i Metropolitans de Barcelona, Sistema d'Indicadors Metropolitans de Barcelona SIMBA, 2019. [ONLINE]. Available: https://iermbdb.uab.cat/ [Accessed: 10- Mar- 2019]

[3] Comunitat de Wikipedia, Front and back ends, 2019. [ONLINE]. Available: https://en.wikipedia.org/wiki/Front_and_back_ends [Accessed: 10- Mar- 2019]

[4] Diego Lázaro, Introducción a REST en PHP, 2019. [ONLINE]. Available: https://diego.com.es/introduccion-a-rest-en-php [Accessed: 10- Mar- 2019]

[5] Comunitat de Wikipedia, Application programming interface, 2019. [ONLINE]. Available: https://en.wikipedia.org/wiki/Application_programming_interface [Accessed: 10- Mar- 2019]

[6] Comunitat de Wikipedia, CSV, 2017. [ONLINE]. Available: https://ca.wikipedia.org/wiki/CSV [Accessed: 10- Mar- 2019]

[7] Comunitat de Wikipedia, Unified Modeling Language, 2019. [ONLINE]. Available: https://en.wikipedia.org/wiki/Unified_Modeling_Language [Accessed: 04- Apr- 2019]

[8] David Carr, Markus Gray. (31 de juliol de 2018). Beginning PHP: Master the latest features of PHP 7 and fully embrace modern PHP development. Ed: Packt Publishing.

[9] Comunitat de Wikipedia, Apache HTTP Server, 2019. [ONLINE]. Available: https://en.wikipedia.org/wiki/Apache_HTTP_Server [Accessed: 10- Mar- 2019]

[10] Comunitat de Wikipedia, Hypertext Transfer Protocol, 2019. [ONLINE]. Available: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol [Accessed: 10- Mar- 2019]

[11] Comunitat de Wikipedia, PostgreSQL, 2019. [ONLINE]. Available: https://ca.wikipedia.org/wiki/PostgreSQL [Accessed: 17- Mar- 2019]

[12] BBVAOpen4U, API REST: qué es y cuáles son sus ventajas en el desarrollo de proyectos, 2016. [ONLINE]. Available: http://ves.cat/epCY [Accessed: 10- Mar- 2019]

[13] Comunitat de Wikipedia, Uniform Resource Identifier, 2019. [ONLINE]. Available: https://en.wikipedia.org/wiki/Uniform_Resource_Identifier [Accessed: 10- Mar- 2019]

[14] Seldaek, Monolog, 2018. [ONLINE]. Available: https://github.com/Seldaek/monolog [Accessed: 10- Mar- 2019]

[15] PHP-FIG, PSR-3: Logger Interface, 2019. [ONLINE]. Available: https://www.php-fig.org/psr/psr-3/ [Accessed: 10- Mar- 2019]

[16] Comunitat de Wikipedia, Code refactoring, 2019. [ONLINE]. Available: https://en.wikipedia.org/wiki/Code_refactoring [Accessed: 04- Apr- 2019]

[17] Eclipse Foundation, Eclipse PHP Development Tools, 2019. [ONLINE]. Available: https://www.eclipse.org/pdt/ [Accessed: 04- Apr- 2019]

[18] Comunitat de Wikipedia, Factory method pattern, 2019. [ONLINE]. Available: https://en.wikipedia.org/wiki/Factory_method_pattern [Accessed: 04- Apr- 2019]

[19] Sebastian Bergmann, PHPUnit 7, 2019. [ONLINE]. Available: https://phpunit.de/announcements/phpunit-7.html [Accessed: 04- Apr- 2019]

[20] Comunitat de Wikipedia, Singleton pattern, 2019. [ONLINE]. Available: https://en.wikipedia.org/wiki/Singleton_pattern [Accessed: 04- Apr- 2019]

[21] Comunitat de OWASP, HttpOnly, 2017. [ONLINE]. Available: https://www.owasp.org/index.php/HttpOnly [Accessed: 20- May- 2019]

APÈNDIX

A.1 Diagrames de classes

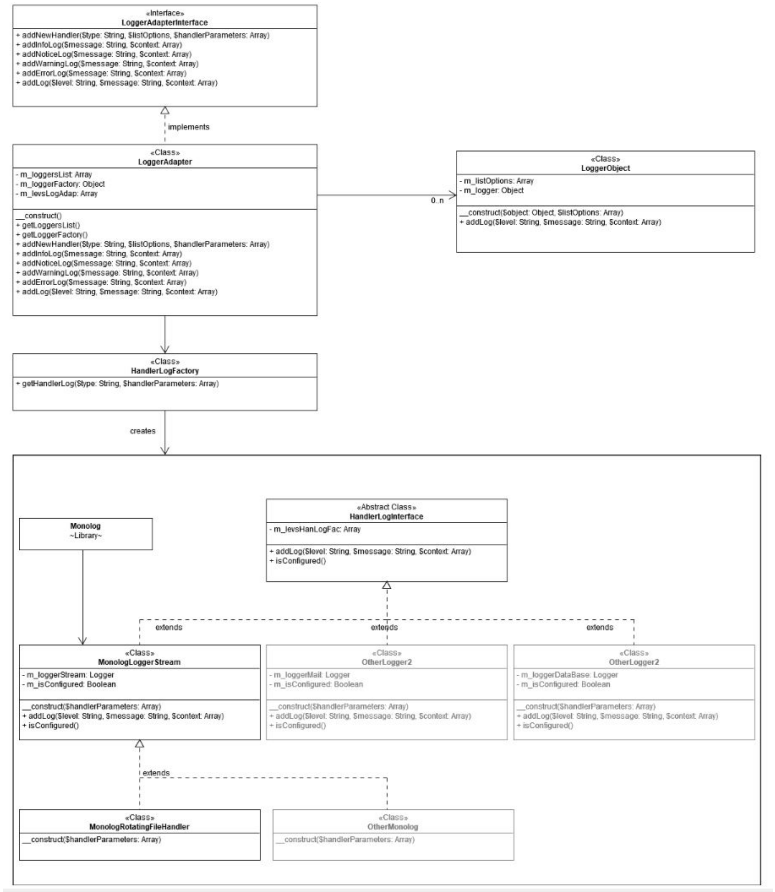

Fig. 3: Estructura interna de la llibreria LoggerAdapter

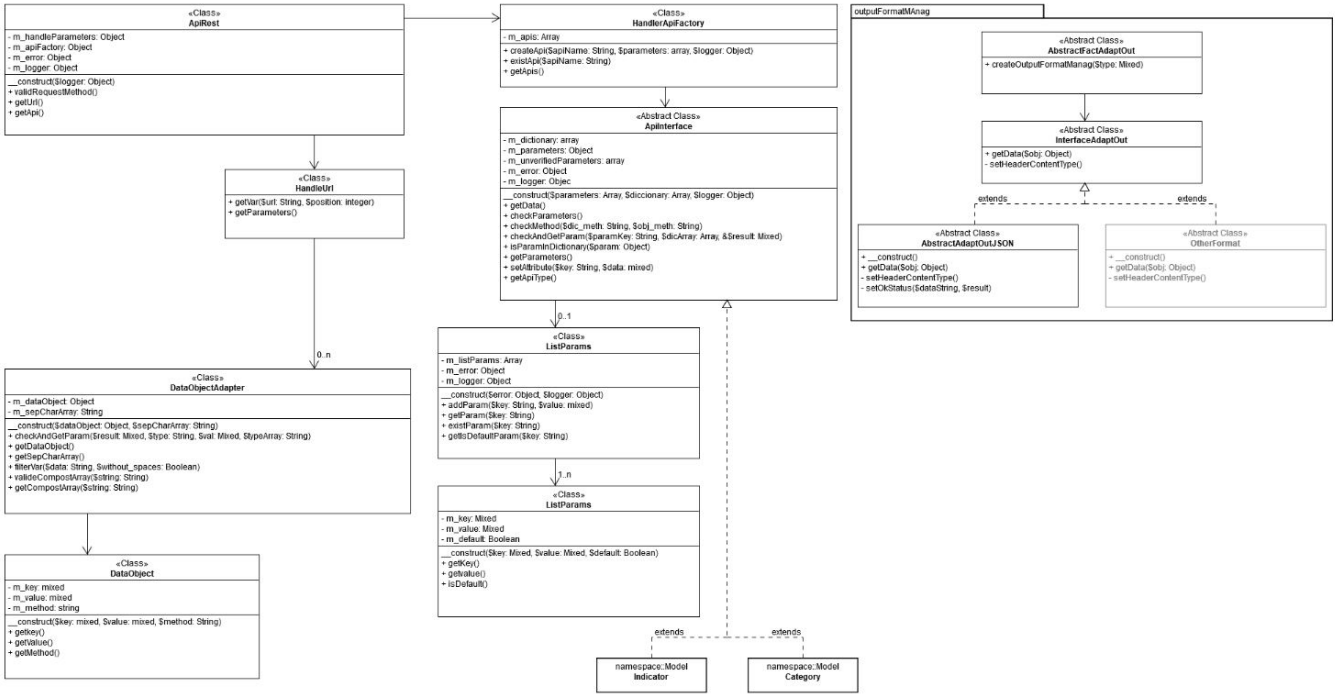

Fig. 4: Estructura interna de classes de la llibreria API REST

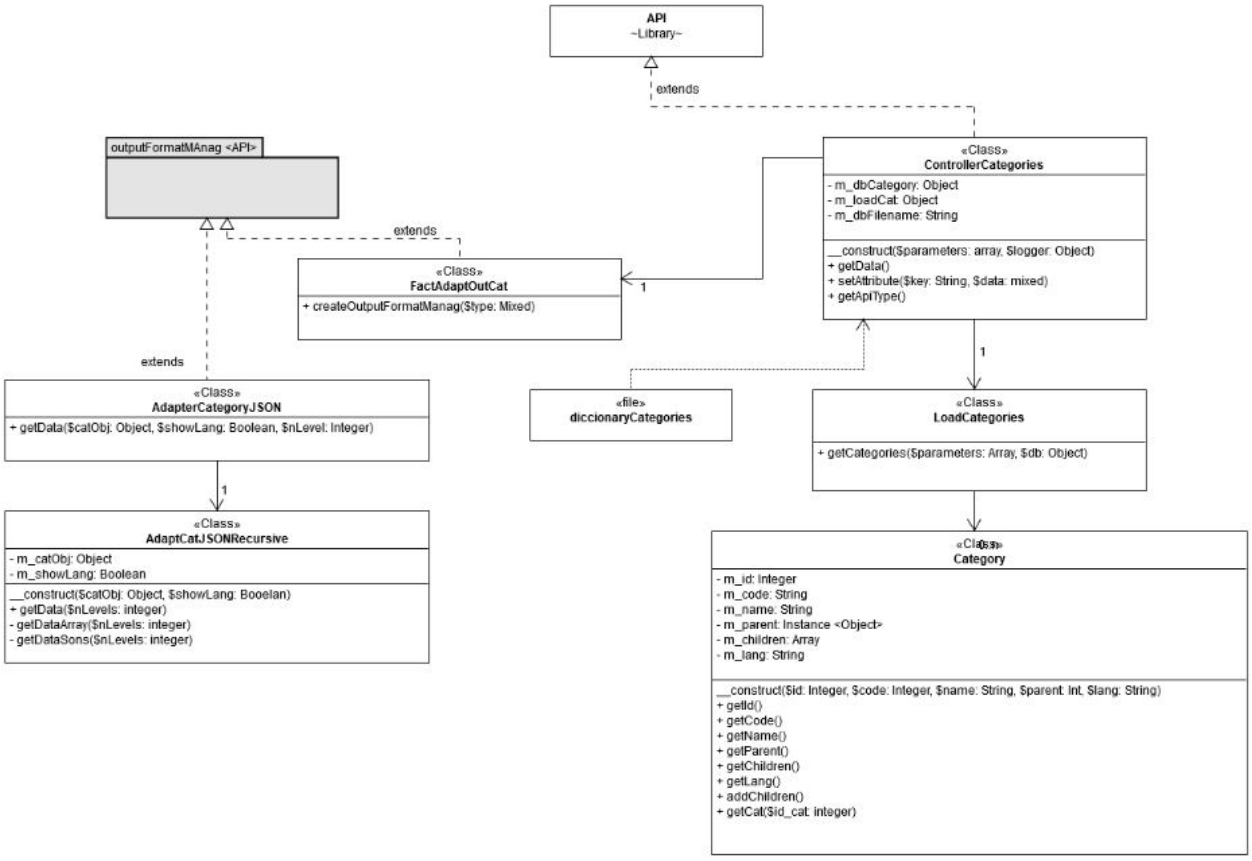

Fig. 5: Estructura interna de l'aplicació Category

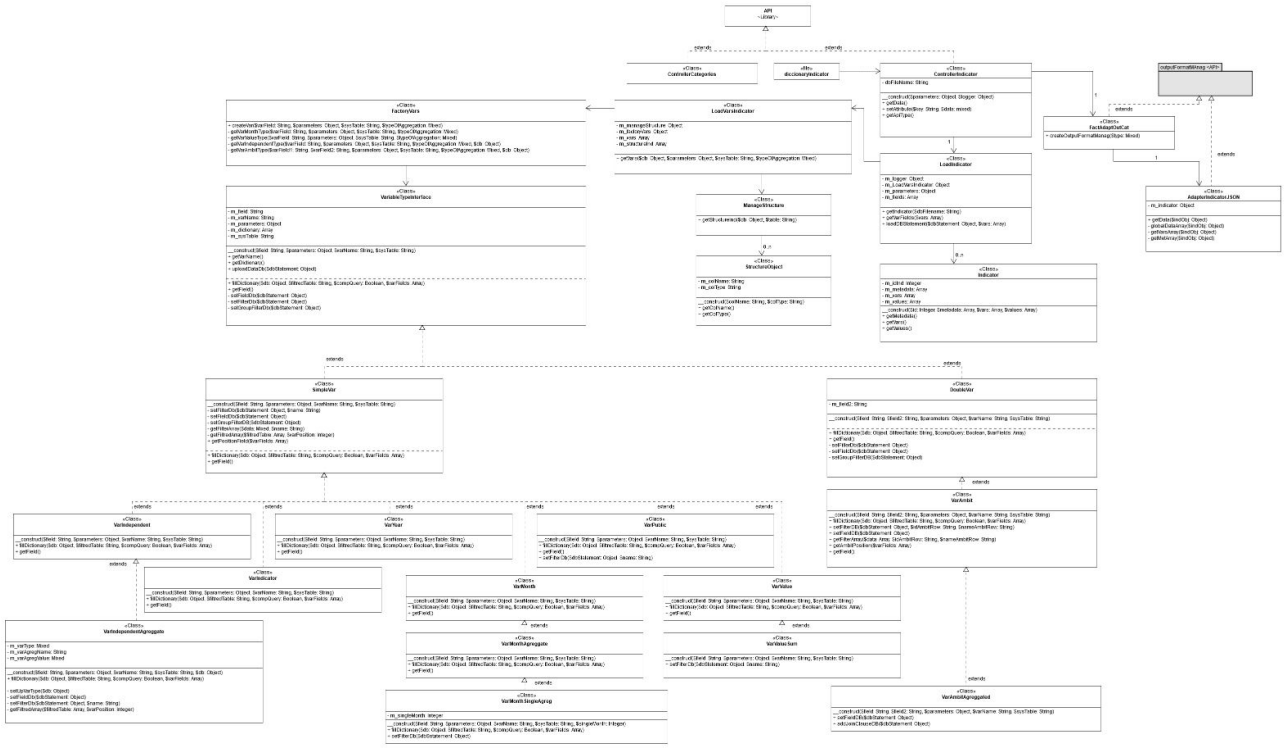

Fig. 6: Estructura interna de l'aplicació Indicator

A.2 Exemple dels resultats de les aplicacions

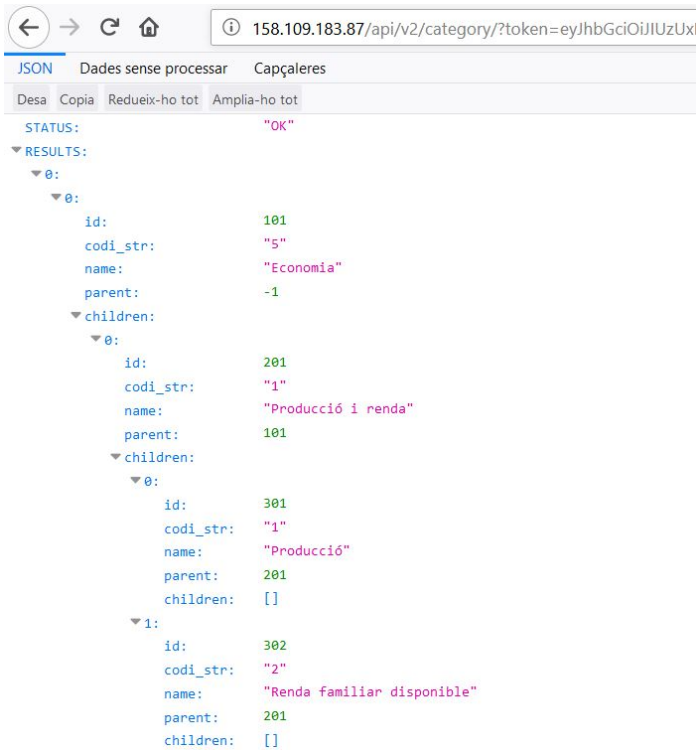

Fig. 7: Resultat de l'execució de l'aplicació Category on es pot observar com, amb l'estàndard JSON, les categories estan organitzades en una estructura jeràrquica de nodes

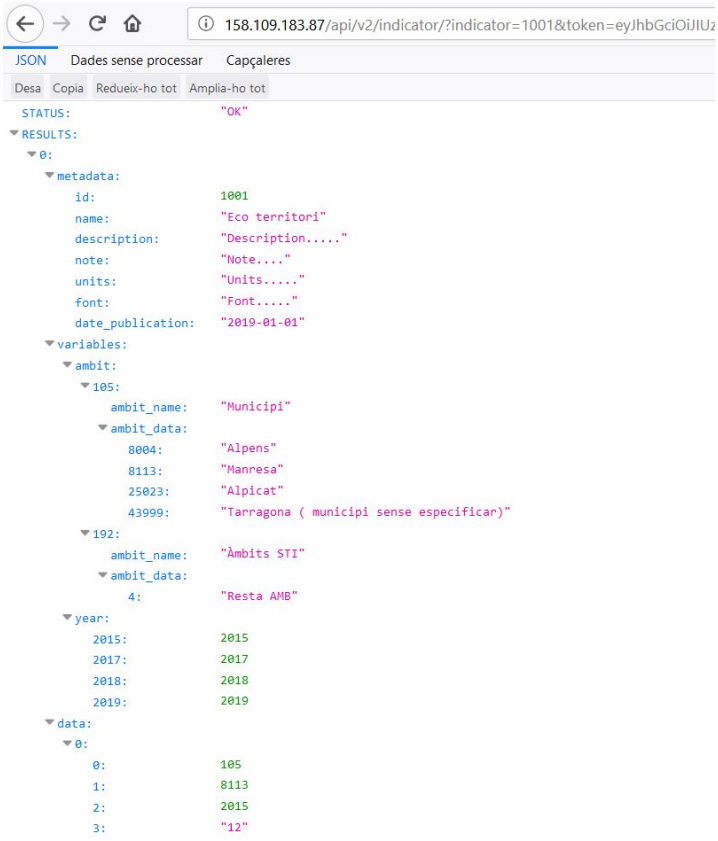

Fig. 8: Resultat de l'execució de l'aplicació Indicator on es pot observar les dades obtingudes amb format JSON de l'indicador 1001 (el número s'especifica a l'URI)